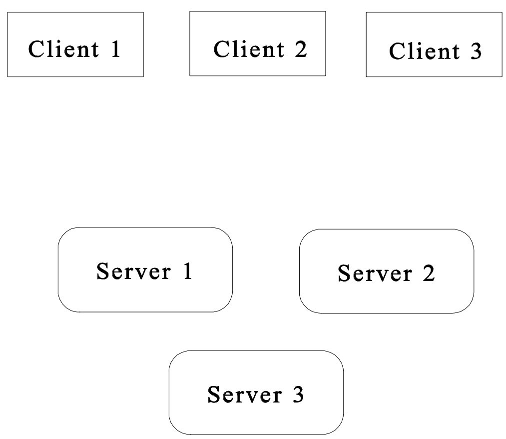
Distributed Data Management

☺ Introduction

- Involves the distribution of data and work among more than one machine in the network.

- Distributed computing is more broad than canonical client/server, in that *many* machines may be processing work on behalf of a single client.



- Operation:
  1. User requests data from the local host.
  2. The goes out over the network to submit the request for data or service to a remote host.
  3. Remote host processes the request and sends the data or the results back to the local host.
  4. Local host hands the reply to the client, which is unaware that the request was executed by multiple servers.

Distributed Data Management (cont.)

☺ Introduction (cont.)

- Notes:

  1. Distributed computing is generally perceived as a natural extension of the client/server model.

  2. The only conceptual difference is that now the host server (coordinator) orchestrates the distributed computation with its "peers" rather than doing everything on its own.

  3. Any host server can act as a coordinator, depending on where the request originated.

- Benefits

  ✓ Placement of data closer its source.

  ✓ Automatic movement of data to where it is most needed.

  ✓ Placement of data closer to the users (through replication).

  ✓ Higher data availability through data replication.

  ✓ Higher fault tolerance through elimination of a single point of failure.

  ✓ Potentially more efficient data access (higher throughput and greater potential for parallelism).

  ✓ Better scalability w.r.t. the application and users' needs.

Distributed Data Management (cont.)

☺ C.J.Date's Principles

- Summary:

1. Local autonomy.
2. No reliance on a central site.
3. Continuous operation.
4. Location independence.
5. Fragmentation independence.
6. Replication independence.
7. Distributed query processing.
8. Distributed transaction management.
9. Hardware independence.
10. Operating system independence.
11. Network independence.
12. DBMS independence.

Distributed Data Management (cont.)

☺ C.J.Date's Principles (cont.)

- Local autonomy
  - ✓ The sites should be independent of each other *to the maximum possible extent*.
  - ✓ This means:
    1. Each site has its own DBMS operating on the local database(s).
    2. The DBMS autonomously handles the management, integrity, security, concurrency and recovery of its local databases (no other site can do this).
    3. Access to local data requires only the local resources.
    4. A local site must cooperate with other sites in order to access remote data.
  - ✓ Violation of this principle would require all sites to be operational all of the time, which is impossible.
- No reliance on a central site
  - ✓ All sites must be treated as equal; no site is more important than any other.
  - ✓ This means, there is no such site which is dedicated to act as a coordinator in query processing, transaction management, security management, etc.

Distributed Data Management (cont.)

☺ C.J.Date's Principles (cont.)

- No reliance on a central site (cont.)
  - ✓ This does not mean that individual requests (or transactions) cannot have central point of control; only that the point of control can be anywhere in the system.
  - ✓ Violation of this principle leads to the creation of a bottleneck and decreased fault tolerance of the system.
- Continuous operation
  - ✓ No planned database activity, e.g administration and recovery, should require system shutdown.
  - ✓ It should be possible to execute all such activities while the system and its databases are "on-line".
  - ✓ If this is not satisfied, availability of data is reduced.
  - ✓ To achieve the principle, the system requires:
    - On-line backup;
    - On-line check and repair functions;
    - On-line transaction recovery;
    - Increased fault tolerance through perhaps disk mirroring and "switch over to alternate site".

Distributed Data Management (cont.)

☺ C.J.Date's Principles (cont.)

■ Location independence

✓ Also referred to as *location transparency* (natural extension of the principle of *data independence*).

✓ Users and applications should not know where the data is physically stored.

✓ An application should behave, at least conceptually, as if all data were stored at the local host.

✓ Desirable for two reasons:

1. Simplifies application development.
2. Allows data to migrate from site to site without invalidating application programs and activities.

✓ Violation of this principle leads to:

1. Application dependency on the location of data.
2. Reduced potential for data migration.
3. Different treatment of local and remote data.

✓ To support location independence, the system must have:

1. Service brokerage facilities.
2. Distributed, synchronized data dictionary.

Distributed Data Management (cont.)

☺ C.J.Date's Principles (cont.)

- Fragmentation independence

  - ✓ *Fragmentation*: table can be partitioned horizontally or vertically into fragments distributed across different sites.

  - ✓ Required to increase performance (through parallel execution) and allow data to reside close to its origin.

  - ✓ Regardless of the internal fragmentation, the table must appear to the applications and the user as one whole table.

  - ✓ The system must determine which fragments need to be accessed in order to answer a query.

  - ✓ Closely related to location transparency.

  - ✓ Violating this principle means that the application needs to keep track of where the individual pieces of a table reside, which complicates the application logic.

  - ✓ To support the principle, the system must:

    1. Maintain the fragmentation information in the distributed data dictionary.

    2. Implement the logic of reconstructing a table from its fragments.

    3. Guarantee that updates of individual fragments are subject to global transactional control.

Distributed Data Management (cont.)

☺ C.J.Date's Principles (cont.)

- Replication independence

  ✓ *Replication*: process of creating and automatically maintaining synchronized copies (replicas) of the same item, table, or database on multiple sites.

  ✓ Replication provides greater performance (data is manipulated closer to the user) and increased data availability (an object is available as long as at least one replica is available).

  ✓ According to the principle of replication independence, users and applications should behave, at least conceptually, as if there is only a single copy of the data.

  ✓ Violating the principle means that the responsibility for keeping the replicas synchronized is placed on the user.

  ✓ To adhere to the principle, the system must:

  1. Propagate the updates automatically.

  2. Deal appropriately with potential update conflicts and integrity violations (that may occur on one but not on the other site).

  3. Maintain the transactional properties of the system...

  ✓ Extremely difficult principle to support.

Distributed Data Management (cont.)

☺ C.J.Date's Principles (cont.)

- Distributed query processing

  ✓ The system must be intelligent enough to support the best access path for a given query;

  ✓ Presupposes several things:

  1. Query execution under the global control of the local site which originally received the request (acts as a coordinator).

  2. Requests or parts of requests executed at places where the data resides.

  3. Query optimization which takes into account not only local but also global factors, e.g. network characteristics, node availability, the costs of distributed execution, etc.

  4. Distributed dictionary with up-to-date cost estimates.

  4. Distributed join capability.

  ✓ Without an intelligent distributed query execution, the system may decide to move all records to a single site and perform filtering there, which would increase network traffic and query execution time.

  ✓ Set of extremely difficult problems associated with distributed query optimization and execution.

Distributed Data Management (cont.)

☺ C.J.Date's Principles (cont.)

- Distributed transaction management

  - ✓ Intended to provide atomicity, consistency, integrity, and durability across different portions of a distributed database.

  - ✓ Without the principle, a distributed database may be left in a globally inconsistent state, even though all local portions are internally consistent.

  - ✓ Problems:

    1. Distributed commit and abort.
    2. Distributed locking.
    3. Distributed deadlock detection.
    4. Global logging and recovery.
    5. Global integrity enforcement.
    6. Global administration, security, and control.

  - ✓ Must rely on a coordinator site to orchestrate distributed commit or abort, usually using a two-phase commit protocol.

Distributed Data Management (cont.)

☺ C.J.Date's Principles (cont.)

- Hardware independence
  - ✓ Distributed database system should be able to run on different hardware platforms, each being an equal partner.
  - ✓ In typical network environments, it is unrealistic to expect homogenous computers.
  - ✓ The system should be able to take advantage of massive CPU power (e.g., MPP or SMP machines).
- Operating system independence
  - ✓ The system should not be written for a single operating system (rationale same as for hardware independence).
- Network independence
  - ✓ The system should not depend on particular network implementations and protocols.
- DBMS independence
  - ✓ Often called *database interconnectivity*.
  - ✓ System must cope with legacy data and provide a support for interconnectivity of existing database repositories.
  - ✓ Difficult goal; attracts much research attention.

Distributed Data Management (cont.)

☺ Problems

- Non-exhaustive list:
  - ✓ Object naming;
  - ✓ Query processing;
  - ✓ Data dictionary management;
  - ✓ Fragmentation;
  - ✓ Global transactions;
  - ✓ Global deadlock detection...
- Object naming
  - ✓ Usually, two types of names used: textual names and globally -unique system ids.
  - ✓ *Textual name*: name by which the object is normally referenced in 4GL queries.
    - An object is usually allowed to have aliases.
  - ✓ *Globally-unique system id*: an immutable, system-defined id; persists for as long as the object exists and is carried with the object across the extents in which it may appear.
    - An object id includes a *place-of-origin*, fully qualified aid assigned at the time the object was first created, e.g. <*site id*>@<*database id*>@<*table id*>@<*tuple id*>.

Distributed Data Management (cont.)

☺ Problems (cont.)

- Query processing

  - ✓ Distributed query processing requires a *global optimization*, followed by *local optimizations* at each selected site.

  - ✓ The performance ratio between an unoptimized and optimized version of a query can be as much as days/seconds.

  - ✓ Types of query optimizations: rule-based and cost-based.

  - ✓ *Rule-based optimizer* selects an access plan based on certain pre-defined rules.

    - For example, in a distributed join of two tables, the optimizer may always move the left-most referenced table to the second site and perform the join there.

    - Can be very inefficient due to "reduced intelligence".

  - ✓ *Cost-based optimization* selects access plan based on the estimated cost of query processing.

  - ✓ Relies heavily on various statistics accumulated and maintained by the DBMS in the data dictionary

    - For example, statistics may include the size of each remote table involved in the query, the network speed, the CPU and I/O speed at each site, etc.

Distributed Data Management (cont.)

☺ Problems (cont.)

- Query processing (cont.)

  ✓ Example: server S1 maintains table
  Supplier(Supplier, Address, City)
  and server S2 maintains table
  Supplies(Supplier, Part#).

  Query: "Find name and address of each supplier of part #110 from Chicago".

  1. DBMS on some third site S3 receives the query and becomes the coordinator C for the query.
  2. Using the catalog, C determines that Supplies is remote and modifies the query to reflect the fact.
  3. Based on estimated costs and characteristics of each servers, C decides to perform join at S3.
  4. C instructs DBMS at S1 to select all suppliers from Boston and send the result to S3.
  5. C instructs DBMS at S2 to select the suppliers for part 10 and send the result to S3.
  6. C receives the results from S1 and S2 and performs their join.
  7. C returns the result to the client.

Distributed Data Management (cont.)

☺ Problems (cont.)

■ Data dictionary management

✓ Alternatives: centralized, fully replicated, and partitioned.

✓ *Centralized dictionary*: stored once at a central site.

- Violates the principle of no reliance on a central site.
- Reduced fault tolerance and the dictionary is a bottleneck.

✓ *Fully replicated*: replicated on every site participating in the global database.

- Loss of site autonomy.
- Difficult replica synchronization and update propagation.

✓ *Partitioned*: each site maintains its own local dictionary.

- Global dictionary is a union of all local dictionaries.
- May require many remote accesses to locate an item.

✓ Hybrid approaches frequently applied, e.g.:

- Fully replicated *name table* that contains name or alias to system id mappings.
- Local dictionaries that include:
  (a) An entry for each object *created* at that site; and
  (b) An entry for each object *stored* at that site.
- At most two remote accesses to locate an object!

Distributed Data Management (cont.)

☺ Problems (cont.)

- Fragmentation

  ✓ A *fragment* is any piece of a table obtained from the original table using selection and projection.

  ✓ Ease of fragmentation is the reason why most distributed database systems are relational.

  ✓ Fragments are assumed to be disjoint (i.e., the intersection of any two fragments of a relation is an empty set).

  ✓ Two types of fragmentation: *horizontal* and *vertical*.

  ✓ *Horizontal fragmentation*: partitions the original table by distributing its tuples across different sites.

    - Simple and applied frequently.

    - Reconstruction is done using appropriate operation.

  ✓ *Vertical fragmentation*: partitions each table through a regular lossless-join decomposition.

    - Stores different subsets of its attributes (projections of the original table) at different sites.

    - Reconstruction is done using a join operation.

    - The decomposition must be lossless, so that joining the fragments will indeed reconstruct the original relation.

Distributed Data Management (cont.)

☺ Problems (cont.)

■ Global transactions

✓ Used to ensure that a transaction operating on a distributed database (called *global transaction*) is atomic.

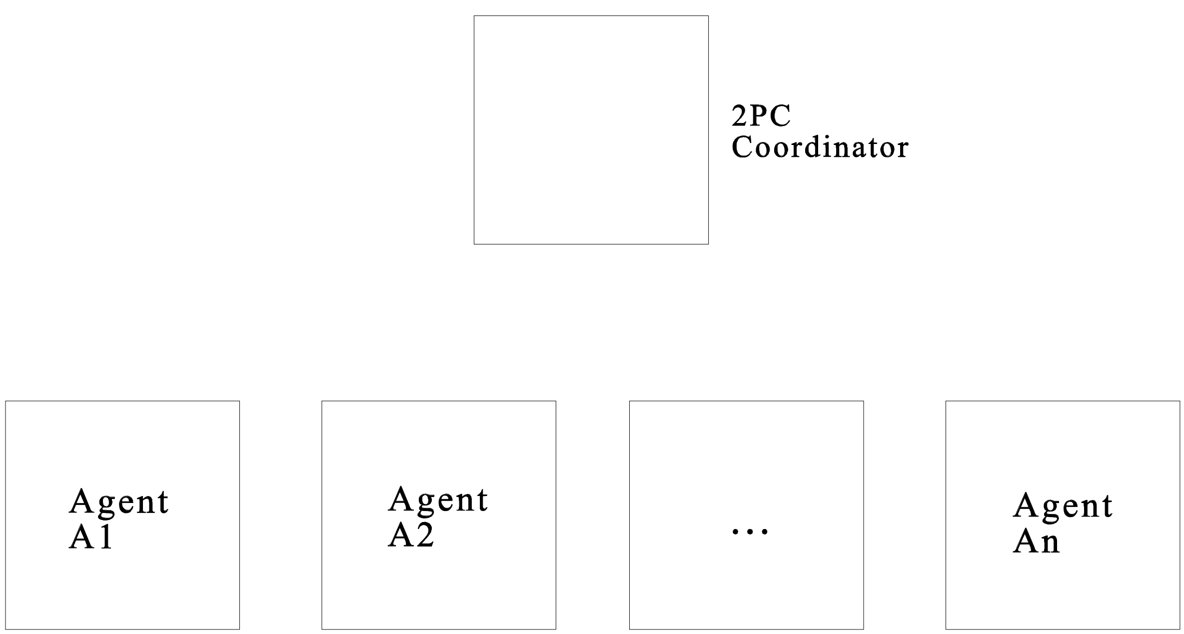
✓ The sites participating in a global transaction must either all commit their work or all must abort it.

✓ Achieved by means of the *two-phase commit protocol*.

✓ Initially:

- The DBMS on the site that first receives the query becomes the coordinator.

- The coordinator dispatches subtasks to the appropriate remote *agents* (DBMSs on remote sites).



Distributed Data Management (cont.)

☺ Problems (cont.)

- Global transactions (cont.)

  ✓ Two-phase commit protocol:

  Phase 1: Starts when the coordinator receives the request to commit:

  a) Coordinator pools each agent, asking for its vote.
  b) If ready, each agent forces log records to its local log.
  c) Assuming the force-write is successful, the agent returns "OK"; otherwise, returns "Not OK".

  Phase 2: Starts when all agents return their votes:

  a) The coordinator makes the decision: if *all* votes are "OK", the decision is "commit"; otherwise, the decision is "abort".
  b) The coordinator forces the log record describing the decision to the log.
  c) The coordinator informs each agent of its decision.
  d) Depending upon the instruction received, each agent commits or aborts its local work for the transaction writing an appropriate record in the log.
  e) If the decision is to commit, each agent acknowledges the commit message; when all acknowledgments arrive, the coordinator writes a "commit completed" record in its log and terminates the transaction.

  ✓ If a failure occurs during the 2PC process:

  - The restart at the coordinator site looks for the decision record at the step 2b and continues as before.

Distributed Data Management (cont.)

☺ Problems (cont.)

- Global deadlocks
  - ✓ Scenario:
    - a) At site S1, the agent of transaction T1 is waiting for the agent of the transaction T2 to release the lock.
    - b) At site S2, the agent of transaction T2 is waiting for the agent of the transaction T1 to release the lock.
  - ✓ Note, neither site can detect the deadlock using only the local resources.
  - ✓ Global deadlock detection requires tremendous communication overhead and resource tracking.
  - ✓ Practical systems usually employ timeouts on lock requests to abort transactions that time out.